

Lecture 3

Introduction to COBOL programming

COBOL's Forte

- COBOL is generally used in situations where the volume of data to be processed is large.
- These systems are sometimes referred to as “**data intensive**” systems.
- Generally, large volumes of data arise **not** because the data is inherently voluminous but because the **same items** of information have been recorded about a **great many instances** of the same object.

A Full COBOL program.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.    SequenceProgram.  
  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01  Num1          PIC 9  VALUE ZEROS.  
01  Num2          PIC 9  VALUE ZEROS.  
01  Result        PIC 99 VALUE ZEROS.  
  
PROCEDURE DIVISION.  
CalculateResult.  
    ACCEPT Num1.  
    ACCEPT Num2.  
    MULTIPLY Num1 BY Num2 GIVING Result.  
    DISPLAY "Result is = ", Result.  
    STOP RUN.
```

The A and B margins of a COBOL program

1	6	7	8	11	7	7	8
				12	2	3	0
		A	B				
		*					
			WORKING-STORAGE SECTION.				
		*					
		77	END-OF-SESSION-SWITCH	PIC X	VALUE "N".		
		77	SALES-AMOUNT	PIC 9(5)V99.			
		77	SALES-TAX	PIC Z,ZZZ.99.			
		*					
			PROCEDURE DIVISION.				
		*					
		000	CALCULATE-SALES-TAX.				
		*					
			PERFORM 100-CALCULATE-ONE-SALES-TAX				
			UNTIL END-OF-SESSION-SWITCH = "Y".				
			DISPLAY "END OF SESSION."				
			STOP RUN.				

The components of each line of code

Columns	Purpose	Remarks
1-6	Sequence	A sequence number that's added when the program is compiled.
7	Indicator	<ul style="list-style-type: none">* treat the line as a <i>comment</i>/ force the program listing to start on a new page- continue code from previous line
8-11	A margin	Some coding elements (like division, section, and procedure names and 77 and 01 level numbers) have to start in this margin.
12-72	B margin	Coding lines that don't start in the A margin have to start in this margin.
73-80	Identification	Not used.

COBOL data description

- Because **COBOL is not typed** it employs a different mechanism for describing the characteristics of the data items in the program.
- COBOL uses what could be described as a “**declaration by example**” strategy.
- In effect, the programmer provides the system with an example, or template, or **PICTURE** of what the data item looks like.
- From the “picture” the system derives the information necessary to allocate it.

Declaring DATA in COBOL

- In COBOL a variable declaration consists of a line containing the following items;
 - ① A level number.
 - ② A data-name or identifier.
 - ③ A PICTURE clause.
- We can give a starting value to variables by means of an extension to the picture clause called the **value clause**.

```
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01  Num1          PIC 999  VALUE ZEROS.  
01  VatRate       PIC V99  VALUE .18.  
01  StudentName   PIC X(10) VALUE SPACES.
```

DATA

Num1	VatRate	StudentName
000	.18	

COBOL 'PICTURE' Clause symbols

- To create the required 'picture' the programmer uses a set of symbols.
- The following symbols are used frequently in picture clauses;
 - 9 (the digit nine) is used to indicate the occurrence of a digit at the corresponding position in the picture.
 - X (the character X) is used to indicate the occurrence of **any** character from the character set at the corresponding position in the picture
 - V (the character V) is used to indicate position of the decimal point in a numeric value! It is often referred to as the “**assumed decimal point**” character.
 - S (the character S) indicates the presence of a sign and can only appear at the beginning of a picture.

COBOL 'PICTURE' Clauses

- Some examples
 - PICTURE 999 a three digit (+ive only) integer
 - PICTURE S999 a three digit (+ive/-ive) integer
 - PICTURE XXXX a four character text item or string
 - PICTURE 99V99 a +ive 'real' in the range 0 to 99.99
 - PICTURE S9V9 a +ive/-ive 'real' in the range ?
- If you wish you can use the abbreviation **PIC**.
- Numeric values can have a maximum of 18 (eighteen) digits (i.e. 9's).
- The limit on string values is usually system-dependent.

Abbreviating recurring symbols

- Recurring symbols can be specified using a ‘repeat’ factor inside round brackets
 - PIC **9(6)** is equivalent to PICTURE **999999**
 - PIC **9(6)V99** is equivalent to PIC **999999V99**
 - PICTURE **X(10)** is equivalent to PIC **XXXXXXXXXXXX**
 - PIC **S9(4)V9(4)** is equivalent to PIC **S9999V9999**
 - PIC **9(18)** is equivalent to PIC **99999999999999999999**

COBOL Literals

- **String/Alphanumeric literals** are enclosed in quotes and may consists of alphanumeric characters

e.g. "Michael Ryan", "-123", "123.45"

- **Numeric literals** may consist of numerals, the decimal point and the plus or minus sign. Numeric literals are not enclosed in quotes.

e.g. 123, 123.45, -256, +2987

Reviewing some of the characters that can be used in Picture clauses

Item type	Characters	Meaning	Examples
Alphanumeric	X	Any character	PIC X
			PIC XXX
			PIC X(3)
Numeric	9	Digit	PIC 99
	S	Sign	PIC S999
	V	Assumed decimal point	PIC S9(5)V99
Numeric edited	9	Digit	PIC 99
	Z	Zero suppressed digit	PIC ZZ9
	,	Inserted comma	PIC ZZZ,ZZZ
	.	Inserted decimal point	PIC ZZ,ZZZ.99
	-	Minus sign if negative	PIC ZZZ,ZZZ-

Picture clauses for alphanumeric items

Value represented	Picture	Data in storage
Y	X	Y
OFF	XXXXX	OFF
714 Main Street	X(20)	714 Main Street

Picture clauses for numeric items

Value represented	Picture	Data in storage	Sign
-26	999V99	02600	(no sign)
+12.50	999V99	01250	(no sign)
+.23	S9(5)V99	0000023	+
-10682.35	S9(5)V99	1068235	-

Picture clauses for numeric edited items

Value represented	Picture	Data in storage
0	Z (4)	(spaces)
0	ZZZ9	0
87	ZZZ9	87
+2,319	ZZ,ZZZ-	2,319
-338	ZZ,ZZZ-	338-
+5,933	Z,ZZZ.99-	5,933.00
-.05	Z,ZZZ.99-	.05-

How to code Picture clauses

- The Picture (Pic) clause defines the format of the data that can be stored in the field.
- When coding a Picture clause, a number in parentheses means that the preceding character is repeated that number of times.
- When data is stored in an alphanumeric item, unused positions to the right are set to blanks.
- When data is stored in a numeric item, unused positions to the left are set to zeros.

The use of literals in Value clauses

Type	Characters	Meaning	Examples
Non-numeric literal	Any	Any character	VALUE "Y" VALUE "END OF SESSION"
Numeric literal	0-9	Digit	VALUE 100
	+ or -	Leading sign	VALUE -100
	.	Decimal point	VALUE +123.55

The use of figurative constants in Value clauses

Type	Constant	Meaning	Examples
Numeric	ZERO	Zero value	VALUE ZERO
	ZEROS		VALUE ZEROS
	ZEROES		VALUE ZEROES
Non-numeric	SPACE	All spaces	VALUE SPACE
	SPACES		VALUE SPACES

Examples of data entries with consistent Picture and Value clauses

Alphanumeric items

77	CUSTOMER-ADDRESS	PIC X(20)	VALUE "213 W. Palm Street".
77	END-OF-FILE-SWITCH	PIC X	VALUE "N".
77	SEPARATOR-LINE	PIC X(20)	VALUE "-----".
77	BLANK-LINE	PIC X(30)	VALUE SPACE.

Numeric items

77	INTEREST-RATE	PIC 99V9	VALUE 12.5.
77	UNIT-COST	PIC 99V999	VALUE 6.35.
77	MINIMUM-BALANCE	PIC S9(5)V99	VALUE +1000.
77	GRAND-TOTAL	PIC S9(5)V99	VALUE ZERO.

How to code Value clauses

- The Value clause defines the value that is stored in the field when the program starts. The value should be consistent with the type of item that's defined by the Picture.
- The characters between the quotation marks in an alphanumeric literal are case sensitive.
- If the Value clause defines a value that is smaller than the field defined by the Picture clause, an alphanumeric field is filled out with spaces on the right; a numeric field is filled out with zeroes on the left.
- If the Value clause defines a value that is larger than can be stored in the field defined by the Picture clause, a compiler error will occur.
- Because a numeric edited item usually receives a value as the result of a Move statement, it usually is not defined with a Value clause.

A Working-Storage Section that contains group items

```
WORKING-STORAGE SECTION.  
*  
01  USER-ENTRIES .  
*  
    05  NUMBER-ENTERED          PIC 9          VALUE 1 .  
    05  INVESTMENT-AMOUNT       PIC 99999 .  
    05  NUMBER-OF-YEARS         PIC 99 .  
    05  YEARLY-INTEREST-RATE    PIC 99V9 .  
*  
01  WORK-FIELDS .  
*  
    05  FUTURE-VALUE            PIC 9(7)V99 .  
    05  YEAR-COUNTER            PIC 99 .  
    05  EDITED-FUTURE-VALUE     PIC Z,ZZZ,ZZZ.99 .  
    05  TODAYS-DATE .  
        10  TODAYS-MONTH        PIC 99 .  
        10  TODAYS-DAY          PIC 99 .  
        10  TODAYS-YEAR         PIC 9(4) .
```

Sections, Paragraphs & Sentences

A Full COBOL program.

```
IDENTIFICATION DIVISION.  
PROGRAM-ID.    SequenceProgram.  
  
DATA DIVISION.  
WORKING-STORAGE SECTION.  
01  Num1          PIC 9  VALUE ZEROS.  
01  Num2          PIC 9  VALUE ZEROS.  
01  Result        PIC 99 VALUE ZEROS.  
  
PROCEDURE DIVISION.  
CalculateResult.  
    ACCEPT Num1.  
    ACCEPT Num2.  
    MULTIPLY Num1 BY Num2 GIVING Result.  
    DISPLAY "Result is = ", Result.  
    STOP RUN.
```

How to code the Procedure Division

- The Procedure Division of a program should be divided into paragraphs.
- Each paragraph in the Procedure Division represents one *procedure* of the program.
- The name of each paragraph can be referred to as either a *paragraph name* or a *procedure name*.
- The name of the first procedure should represent the function of the entire program.
- The names of the procedures called by the first procedure should represent the functions performed by those procedures.

Sections

- A **section** is a block of code made up of one or more **paragraphs**.
- A section begins with the section name and ends where the next section name is encountered or where the program text ends.
- A section name consists of a name devised by the programmer or defined by the language followed by the word SECTION followed by a full stop.
 - SelectUlsterRecords SECTION.
 - FILE SECTION.

Paragraphs

- Each section consists of one or more paragraphs.
- A **paragraph** is a block of code made up of one or more **sentences**.
- A paragraph begins with the paragraph name and ends with the next paragraph or section name or the end of the program text.
- The paragraph name consists of a name devised by the programmer or defined by the language followed by a full stop.
 - PrintFinalTotals.
 - PROGRAM-ID.

Paragraph Example

ProcessRecord.

DISPLAY StudentRecord

READ StudentFile

AT END MOVE HIGH-VALUES TO
StudentRecord

END-READ.

ProduceOutput.

DISPLAY “Here is a message”.

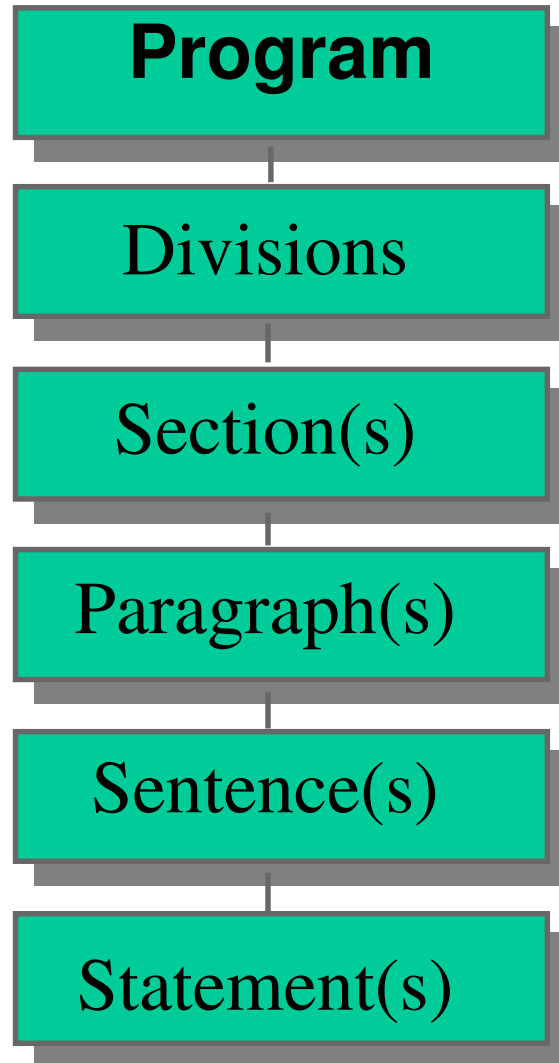
NOTE

The scope of ‘ProcessRecord’ is delimited by the occurrence the paragraph name ‘ProduceOutput’.

Sentences and Statements

- A paragraph consists of one or more sentences.
- A **sentence** consists of one or more **statements** and is terminated by a full stop.
 - MOVE .21 TO VatRate
COMPUTE VatAmount = ProductCost * VatRate.
 - DISPLAY "Enter name " WITH NO ADVANCING
ACCEPT StudentName
DISPLAY "Name entered was " StudentName.
- A **statement** consists of a COBOL **verb** and an **operand** or operands.
 - SUBTRACT Tax FROM GrossPay GIVING NetPay
 - READ StudentFile
AT END SET EndOfFile TO TRUE
END-READ

Structure of COBOL programs



Hierarchical Data Records

Group Items/Records

WORKING-STORAGE SECTION.

01 StudentDetails PIC X(26) .

StudentDetails

H	E	N	N	E	S	S	Y	R	M	9	2	3	0	1	6	5	L	M	5	1	0	5	5	0	F
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Group Items/Records

WORKING-STORAGE SECTION.

01 StudentDetails.

02 StudentName PIC X(10).
02 StudentId PIC 9(7).
02 CourseCode PIC X(4).
02 Grant PIC 9(4).
02 Gender PIC X.

StudentDetails

H	E	N	N	E	S	S	Y	R	M	9	2	3	0	1	6	5	L	M	5	1	0	5	5	0	F
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

StudentName

StudentId

CourseCode

Grant

Gender

Group Items/Records

WORKING-STORAGE SECTION.

01 StudentDetails.

02 StudentName.

03 Surname PIC X(8).

03 Initials PIC XX.

02 StudentId PIC 9(7).

02 CourseCode PIC X(4).

02 Grant PIC 9(4).

02 Gender PIC X.

StudentDetails

H	E	N	N	E	S	S	Y	R	M	9	2	3	0	1	6	5	L	M	5	1	0	5	5	0	F
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

StudentName

StudentId

CourseCode

Grant

Gender

Surname

Initials

LEVEL Numbers express DATA hierarchy

- In COBOL, level numbers are used to decompose a structure into it's constituent parts.
- In this hierarchical structure the higher the level number, the lower the item is in the hierarchy. At the lowest level the data is completely atomic.
- The level numbers 01 through 49 are general level numbers but there are also special level numbers such as 66, 77 and 88.
- In a hierarchical data description what is important is the relationship of the level numbers to one another, not the actual level numbers used.

```
01 StudentDetails.  
  02 StudentName.  
    03 Surname      PIC X(8) .  
    03 Initials     PIC XX .  
  02 StudentId      PIC 9(7) .  
  02 CourseCode     PIC X(4) .  
  02 Grant          PIC 9(4) .  
  02 Gender         PIC X .
```

=

```
01 StudentDetails.  
  05 StudentName.  
    10 Surname      PIC X(8) .  
    10 Initials     PIC XX .  
  05 StudentId      PIC 9(7) .  
  05 CourseCode     PIC X(4) .  
  05 Grant          PIC 9(4) .  
  05 Gender         PIC X .
```


Group and elementary items

- In COBOL the term “**group item**” is used to describe a data item which has been further subdivided.
 - A Group item is declared using a level number and a data name. It cannot have a picture clause.
 - Where a group item is the highest item in a data hierarchy it is referred to as a **record** and uses the level number **01**.
- The term “**elementary item**” is used to describe data items which are atomic; that is, not further subdivided.
- An elementary item declaration consists of;
 - a level number,
 - a data name
 - picture clause.

An elementary item **must** have a picture clause.
- Every group or elementary item declaration **must** be followed by a full stop.

How to code group items

- Use the level numbers 01 through 49.
- Level 01 items must begin in the A margin. Other level numbers can begin in either the A or B margin.
- Whenever one data item has higher level numbers beneath it, it is a *group item* and the items beneath it are *elementary items*.
- You can't code a Picture clause for a group item, but you have to code a Picture clause for an elementary item.
- A group item is always treated as an alphanumeric item, no matter how the elementary items beneath it are defined.
- To make the structure of the data items easy to read and understand, you should align the levels.

Assignments - The Move Statement

The syntax of the Move statement

`MOVE {data-name-1 | literal} TO data-name-2`

Examples of Move statements

`MOVE "Y" TO END-OF-SESSION-SWITCH.`

`MOVE 1 TO PAGE-NUMBER.`

`MOVE NUMBER-ENTERED TO EDITED-NUMBER-ENTERED.`

Legal and illegal moves

Type of move	Legal?
Alphanumeric to alphanumeric	Yes
Numeric to numeric	Yes
Numeric to numeric edited	Yes
Alphanumeric to numeric	Only if the sending field is an unsigned integer
Alphanumeric to numeric edited	Only if the sending field is an unsigned integer
Numeric to alphanumeric	Only if the sending field is an unsigned integer

MOVEing Data

```
MOVE "RYAN" TO Surname.  
MOVE "FITZPATRICK" TO Surname.
```

```
01 Surname    PIC X(8) .
```

C	O	U	G	H	L	A	N
---	---	---	---	---	---	---	---

MOVEing Data

```
MOVE "RYAN" TO Surname.  
MOVE "FITZPATRICK" TO Surname.
```

```
01 Surname    PIC X(8) .
```

R	Y	A	N				
---	---	---	---	--	--	--	--

MOVEing Data

```
MOVE "RYAN" TO Surname.  
MOVE "FITZPATRICK" TO Surname.
```

```
01 Surname    PIC X(8) .
```

F	I	T	Z	P	A	T	R
---	---	---	---	---	---	---	---

 I C K

MOVEing to a numeric item.

- When the destination item is numeric, or edited numeric, then data is aligned along the **decimal point** with zero filling or truncation as necessary.
- When the decimal point is not explicitly specified in either the source or destination items, the item is treated as if it had an assumed decimal point immediately after its rightmost character.

MOVE Examples

```
01 GrossPay          PIC 9(4)V99.
```

```
MOVE ZEROS TO GrossPay.
```

```
MOVE 12.4 TO GrossPay.
```

```
MOVE 123.456 TO GrossPay.
```

```
MOVE 12345.757 TO GrossPay.
```

GrossPay

0	0	0	0	0	0
---	---	---	---	---	---



GrossPay

•

0	0	1	2	4	0
---	---	---	---	---	---



GrossPay

•

0	1	2	3	4	5
---	---	---	---	---	---

6



GrossPay

•

1	2	3	4	5	7	5	7
---	---	---	---	---	---	---	---



•

Examples of numeric to numeric edited moves

Picture of sending field	Data in sending field	Sign of sending field	Picture of receiving field	Edited result
S9(6)	000123	+	ZZZ,ZZ9-	123
S9(6)	012345	-	ZZZ,ZZ9-	12,345-
S9(6)	000000	(no sign)	ZZZ,ZZ9-	0
S9(4)V99	012345	+	ZZZZ.99	123.45
S9(4)V99	000000	(no sign)	ZZZZ.99	.00

Examples of truncation

Picture of sending field	Data in sending field	Picture of receiving field	Edited result
X(3)	Yes	X	Y
S9(6)	012345	S9(3)	345

How to code Move statements

- The Move statement moves data from a literal or a sending field to a receiving field. However, the original data is retained in the sending field.
- If the sending field is a numeric item and the receiving field is numeric edited, the Move statement converts the data from one form to the other.
- If the receiving field is larger than the sending field, the receiving field is filled out with trailing blanks in an alphanumeric move or leading zeros in a numeric move.
- If the receiving field is smaller than the sending field, the data that's moved may be truncated.

Conditional Statements

The syntax of the If statement

```
IF condition
    statement-group-1
[ELSE
    statement-group-2]
[END-IF]
```

The syntax of a simple condition

```
{data-name-1 | literal} relational-operator {data-name-2 | literal}
```

The relational operators

Operator	Meaning	Typical conditions
>	Greater than	NUMBER-ENTERED > ZERO
<	Less than	999 < NUMBER-ENTERED
=	Equal to	END-OF-SESSION-SWITCH = "Y"
>=	Greater than or equal to	LINE-COUNT >= LINES-ON-PAGE
<=	Less than or equal to	SALES-THIS-YEAR <= SALES-LAST-YEAR
NOT	The negative of the operator that follows	NUMBER-ENTERED NOT = 0

If statements without Else and End-If clauses

```
IF SALES-AMOUNT = ZERO
    MOVE "Y" TO END-OF-SESSION-SWITCH.

IF SALES-AMOUNT NOT = ZERO
    COMPUTE SALES-TAX ROUNDED = SALES-AMOUNT * .0785
    DISPLAY "SALES TAX = " SALES-TAX.
```

An If statement with Else and End-If clauses

```
IF SALES-AMOUNT = ZERO
    MOVE "Y" TO END-OF-SESSION-SWITCH
ELSE
    COMPUTE SALES-TAX ROUNDED = SALES-AMOUNT * .0785
    DISPLAY "SALES TAX = " SALES-TAX
END-IF.
```

Nested If statements

```
IF SALES-AMOUNT >= 10000
    IF SALES-AMOUNT < 50000
        COMPUTE SALES-COMMISSION = SALES * COMMISSION-RATE-1
    ELSE
        COMPUTE SALES-COMMISSION = SALES * COMMISSION-RATE-2
    END-IF
END-IF.
```

How to code If statements

- The If statement executes one group of statements if the condition it contains is true, another group of statements if the condition is false and the Else clause is coded.
- When you code an If statement within an If statement, you are coding *nested If statements*.
- The End-If *delimiter* can be used to mark the end of an If statement.

The syntax of the Perform statement

PERFORM procedure-name

An example of a Perform statement

PERFORM 100-GET-USER-ENTRIES.

The operation of the Perform statement

- The Perform statement skips to the procedure that's named and executes the statements in that procedure. Then, it returns to the statement after the Perform, and the program continues.

The syntax of the Perform Until statement

```
PERFORM procedure-name  
    UNTIL condition
```

An example of a Perform Until statement

```
PERFORM 100-CALCULATE-ONE-SALES-TAX  
    UNTIL END-OF-SESSION-SWITCH = "Y".
```

The operation of the Perform Until statement

- The Perform Until statement tests if a condition is true. If it isn't true, the statement performs the procedure that it names until the condition becomes true. Then, the program continues with the statement after the Perform Until statement.
- The execution of a Perform Until statement is often referred to as a *processing loop*, or simply a *loop*.
- The condition in a Perform Until statement is formed the same way it is formed in an If statement.

Defining Condition Names.

$$88 \text{ ConditionName } \left\{ \begin{array}{l} \underline{\text{VALUE}} \\ \underline{\text{VALUES}} \end{array} \right\} \left\{ \begin{array}{l} \text{Literal} \\ \text{LowValue } \left\{ \begin{array}{l} \underline{\text{THROUGH}} \\ \underline{\text{THRU}} \end{array} \right\} \text{HighValue} \end{array} \right\} \dots$$

- Condition Names are defined in the DATA DIVISION using the special level number 88.
- They are always associated with a data item and are defined immediately after the definition of the data item.
- A condition name takes the value **TRUE** or **FALSE** depending on the value in its associated data item.
- A Condition Name may be associated with **ANY** data item whether it is a group or an elementary item.
- The VALUE clause is used to **identify** the values which make the Condition Name TRUE.

```

01 CityCode          PIC 9 VALUE 5.
   88 Dublin         VALUE 1.
   88 Limerick        VALUE 2.
   88 Cork            VALUE 3.
   88 Galway          VALUE 4.
   88 Sligo           VALUE 5.
   88 Waterford       VALUE 6.
   88 UniversityCity VALUE 1 THRU 4.

```

```

IF Limerick
    DISPLAY "Hey, we're home."
END-IF
IF UniversityCity
    PERFORM CalcRentSurcharge
END-IF

```

City Code

5

Dublin	FALSE
Limerick	FALSE
Cork	FALSE
Galway	FALSE
Sligo	TRUE
Waterford	FALSE
UniversityCity	FALSE

```

01 CityCode          PIC 9 VALUE 5.
   88 Dublin         VALUE 1.
   88 Limerick        VALUE 2.
   88 Cork            VALUE 3.
   88 Galway          VALUE 4.
   88 Sligo           VALUE 5.
   88 Waterford       VALUE 6.
   88 UniversityCity VALUE 1 THRU 4.

```

```

IF Limerick
    DISPLAY "Hey, we're home."
END-IF
IF UniversityCity
    PERFORM CalcRentSurcharge
END-IF

```

City Code

2

Dublin	FALSE
Limerick	TRUE
Cork	FALSE
Galway	FALSE
Sligo	FALSE
Waterford	FALSE
UniversityCity	TRUE

```

01 CityCode          PIC 9 VALUE 5.
   88 Dublin          VALUE 1.
   88 Limerick         VALUE 2.
   88 Cork             VALUE 3.
   88 Galway           VALUE 4.
   88 Sligo            VALUE 5.
   88 Waterford        VALUE 6.
   88 UniversityCity  VALUE 1 THRU 4.

```

```

IF Limerick
    DISPLAY "Hey, we're home."
END-IF
IF UniversityCity
    PERFORM CalcRentSurcharge
END-IF

```

City Code

6

Dublin	FALSE
Limerick	FALSE
Cork	FALSE
Galway	FALSE
Sligo	FALSE
Waterford	TRUE
UniversityCity	FALSE

```
01 EndOfFileFlag      PIC 9 VALUE 0.  
88 EndOfFile          VALUE 1.
```

EndOfFileFlag

0

EndOfFile

```
READ InFile  
    AT END MOVE 1 TO EndOfFileFlag  
END-READ  
PERFORM UNTIL EndOfFile  
    Statements  
    READ InFile  
        AT END MOVE 1 TO EndOfFileFlag  
    END-READ  
END-PERFORM
```

```
01 EndOfFileFlag      PIC 9 VALUE 0.  
88 EndOfFile          VALUE 1.
```

EndOfFileFlag

1

EndOfFile

```
READ InFile  
    AT END MOVE 1 TO EndOfFileFlag  
END-READ  
PERFORM UNTIL EndOfFile  
    Statements  
    READ InFile  
        AT END MOVE 1 TO EndOfFileFlag  
    END-READ  
END-PERFORM
```


Arithmetic

The syntax of the Add statement

Format 1

```
ADD {data-name-1 | literal} TO data-name-2 [ROUNDED]  
    [ON SIZE ERROR statement-group]
```

Format 2

```
ADD {data-name-1 | literal-1} {data-name-2 | literal-2} ...  
    GIVING data-name-3 [ROUNDED]  
    [ON SIZE ERROR statement-group]
```

Examples of Add statements

Format 1

```
ADD 1 TO YEAR-COUNTER.  
ADD CUSTOMER-SALES TO GRAND-TOTAL-SALES.
```

Format 2

```
ADD OLD-BALANCE NEW-CHARGES  
    GIVING NEW-BALANCE.  
ADD JAN-SALES FEB-SALES MAR-SALES  
    GIVING FIRST-QUARTER-SALES.
```

Arithmetic Verb Template

$$\text{VERB } \left\{ \begin{array}{l} \text{Identifier} \\ \text{Literal} \end{array} \right\} \left\{ \begin{array}{l} \text{TO} \\ \text{FROM} \\ \text{BY} \\ \text{INTO} \end{array} \right\} \left\{ \begin{array}{l} \text{Identifier} \dots \\ \text{Identifier} \text{ GIVING Identifier} \dots \end{array} \right\} [\text{ROUNDED}]$$

[ON SIZE ERROR StatementB lock END - VERB]

Most COBOL arithmetic verbs conform to the template above.

For example;

ADD Takings TO CashTotal.

ADD Males TO Females GIVING TotalStudents.

SUBTRACT Tax FROM GrossPay.

SUBTRACT Tax FROM GrossPay GIVING NetPay.

DIVIDE Total BY Members GIVING MemberAverage.

DIVIDE Members INTO Total GIVING MemberAverage.

MULTIPLY 10 BY Magnitude.

MULTIPLY Members BY Subs GIVING TotalSubs.

- The exceptions are the **COMPUTE** and the **DIVIDE** with **REMAINDER**.

ADD Examples

ADD Cash TO Total.

Before	3	1000
After	3	1003

ADD Cash, 20 TO Total, Wage.

Before	3	1000	100
After	3	1023	123

ADD Cash, Total GIVING Result.

Before	3	1000	0015
After	3	1000	1003

ADD Males TO Females GIVING TotalStudents.

Before	1500	0625	1234
After	1500	0625	2125

The syntax of the Compute statement

```
COMPUTE data-name [ROUNDED] = arithmetic-expression  
      [ON SIZE ERROR statement-group]
```

The arithmetic operators

- + Addition
- Subtraction
- * Multiplication
- / Division
- ** Exponentiation

Examples of Compute statements

```
COMPUTE YEAR-COUNTER = YEAR-COUNTER + 1
```

```
COMPUTE SALES-TAX ROUNDED =
```

```
    SALES-AMOUNT * .0785.
```

```
COMPUTE SALES-CHANGE = THIS-YEAR-SALES - LAST-YEAR-SALES.
```

```
COMPUTE CHANGE-PERCENT ROUNDED =
```

```
    SALES-CHANGE / LAST-YEAR-SALES * 100
```

```
ON SIZE ERROR
```

```
    DISPLAY "SIZE ERROR ON CHANGE PERCENT".
```

```
COMPUTE NUMBER-SQUARED = NUMBER-ENTERED ** 2
```

ROUNDING

Receiving Field	Actual Result	Truncated Result	Rounded Result
PIC 9(3)V9.	123.25	123.2	123.3
PIC 9(3).	123.25	123	123

- ◆ The **ROUNDED** option takes effect when, after decimal point alignment, the result calculated must be truncated on the right hand side.
- ◆ The option adds 1 to the receiving item when the leftmost truncated digit has an absolute value of 5 or greater.

The ON SIZE ERROR option

Receiving Field	Actual Result	SIZE ERROR
PIC 9(3)V9.	245.96	Yes
PIC 9(3)V9.	1245.9	Yes
PIC 9(3).	124	No
PIC 9(3).	1246	Yes
PIC 9(3)V9 Not Rounded	124.45	Yes
PIC 9(3)V9 Rounded	124.45	No
PIC 9(3)V9 Rounded	3124.45	Yes

- ◆ A size error condition exists when, after decimal point alignment, the result is truncated on either the left or the right hand side.
- ◆ If an arithmetic statement has a rounded phrase then a size error only occurs if there is truncation on the left hand side (most significant digits).

Examples of Compute statements

Statement	A (After) S9 (3) V9	A (Before) S9 (3) V9	B S9 (3)	C S9 (3)
COMPUTE A = A + B	5.0	2.0	3	
COMPUTE A = A + 1	3.0	2.0	1	
COMPUTE A ROUNDED = B / C	.3	?	1	3
COMPUTE A = B / C * 100	66.6	?	2	3
COMPUTE A ROUNDED = B / C * 100	66.7	?	2	3
COMPUTE A = 200 / B - C	37.0	?	5	3
COMPUTE A = 200 / (B - C)	100.0	?	5	3
COMPUTE A = 10 ** (B - C)	Size Error	?	5	1
COMPUTE A = A + (A * .1)	110.0	100.0		
COMPUTE A = A + 1.1	110.0	100.0		

Accept & Display Statements

How the Accept statement works

- When the Accept statement is run, the computer waits for the user to type an entry on the keyboard and press the Enter key.
- When the user presses the Enter key, the entry is stored in the variable identified on the Accept statement, and the cursor moves to the next line on the screen.
- The user entry should be consistent with the Picture of the variable. If it isn't, it will be truncated or adjusted.

Mainframe notes

- On an IBM mainframe, the Accept statement gets its data from the SYSIN device. As a result, this device must be set to the terminal keyboard.
- You have to enter the data more precisely when you use a mainframe than you do when you use a PC.

The syntax of the Display statement

```
DISPLAY {data-name-1 | literal-1} ...
```

Examples of Display statements

```
DISPLAY " ".  
DISPLAY 15000.  
DISPLAY "-----".  
DISPLAY "End of session."  
DISPLAY SALES-AMOUNT.  
DISPLAY "THE SALES AMOUNT IS " SALES-AMOUNT ".".  
DISPLAY "THE SALES TAX ON " SALES-TAX ".".
```

The data displayed by the statements above

(one space or a blank line)

15000

End of session.

100.00

THE SALES AMOUNT IS 100.00.

THE SALES TAX IS 7.85.

How to code Display statements

- The Display statement displays one or more literal or variable values on the screen of a monitor or terminal. After it displays these values, the cursor moves to the next line on the screen.
- After the word DISPLAY, you can code one or more literals or variable names.
- If you code more than one literal or variable name after the word DISPLAY, you must separate them by one or more spaces.

Mainframe note

- On an IBM mainframe, the Display statement sends its data to the SYSOUT device. As a result, this device must be set to the terminal screen.

The syntax of the Accept statement

ACCEPT data-name

An example of an Accept statement

ACCEPT SALES-AMOUNT.

The operation of some typical Accept statements

Picture	User entry	Data stored	Notes
S999	10	10	
S999	787	787	
S999	-10	-10	
S999	5231	231	Truncated on the left
999	-100	100	Sign dropped
9(3)V99	458.12	458.12	
9(3)V99	45812	812.00	Truncated on the left
9(3)V99	4735.26	735.26	Truncated on the left
X	Y	Y	
X	Yes	Y	Truncated on the right

Getting the Date & Time

The syntax of the Current-Date function

FUNCTION CURRENT-DATE

The data description for the data returned by the Current-Date function

```
01  CURRENT-DATE-AND-TIME .  
    05  CD-YEAR                PIC 9(4) .  
    05  CD-MONTH              PIC 9(2) .  
    05  CD-DAY                PIC 9(2) .  
    05  CD-HOURS              PIC 9(2) .  
    05  CD-MINUTES            PIC 9(2) .  
    05  CD-SECONDS            PIC 9(2) .  
    05  CD-HUNDREDTH-SECONDS PIC 9(2) .  
    05  CD-GREENWICH-MEAN-TIME-SHHMM PIC X(5)
```

Two statements that use the Current-Date function

DISPLAY FUNCTION CURRENT-DATE.

MOVE FUNCTION CURRENT-DATE TO CURRENT-DATE-AND-TIME.

How to use the Current-Date function

- In 1989, an addendum was added to the COBOL-85 standards that provided for *intrinsic functions*. Most modern COBOL-85 compilers include these functions, and they were made standard in COBOL-2002.
- The Current-Date function gets the current date and time. It can be used to represent a single variable in any statement where that makes sense.
- Before the Current-Date function became available, you had to use the Accept Date and Accept Time statements to get the date and time.
- The five-character Greenwich Mean Time that is returned by this function indicates the number of hours and minutes that the current time is ahead or behind Greenwich Mean Time.

The syntax of the Accept statement for getting the date and time

```
ACCEPT data-name FROM DATE [YYYYMMDD]
```

```
ACCEPT data-name FROM TIME
```

Description

- If your compiler doesn't support the Current-Date function, you need to use the Accept Date and Accept Time statements to get the date and time.
- With the VS COBOL II compiler, you can't get the date with a four-digit year because the YYYYMMDD option isn't available.

IBM mainframe code that gets the time

```
01  CURRENT-DATE-AND-TIME .  
    05  CD-CURRENT-TIME .  
        10  CD-CURRENT-HOURS          PIC 99 .  
        10  CD-CURRENT-MINUTES        PIC 99 .  
        10  CD-CURRENT-SECONDS        PIC 99 .  
        10  CD-CURRENT-HUNDREDTHS     PIC 99 .  
  
    .  
    .  
    ACCEPT CD-CURRENT-TIME FROM TIME .
```

Describing DATA

Basic coding rules

- You can use capital or lowercase letters when you code a COBOL program.
- Double quotes (") are required for quotation marks by most compilers.
- Single quotes (') are commonly used on most mainframe compilers, although this can be changed by a compiler option.
- One space is treated the same as any number of spaces in sequence. As a result, you can code more than one space whenever you want to indent or align portions of code.

The rules for forming a program name in standard COBOL

- Use letters, the digits 0 through 9, and the hyphen.
- Don't start or end the name with a hyphen.
- Use a maximum of 30 characters.

The rules for forming a program name on a mainframe compiler

- Start the name with a letter.
- Use letters and digits only.
- Use a maximum of eight characters.

The rules for forming a data name

- Use letters, the digits 0 through 9, and hyphens only.
- Don't start or end the name with a hyphen.
- Use a maximum of 30 characters.
- Use at least one letter in the name.
- Don't use the same name as a COBOL *reserved word*.